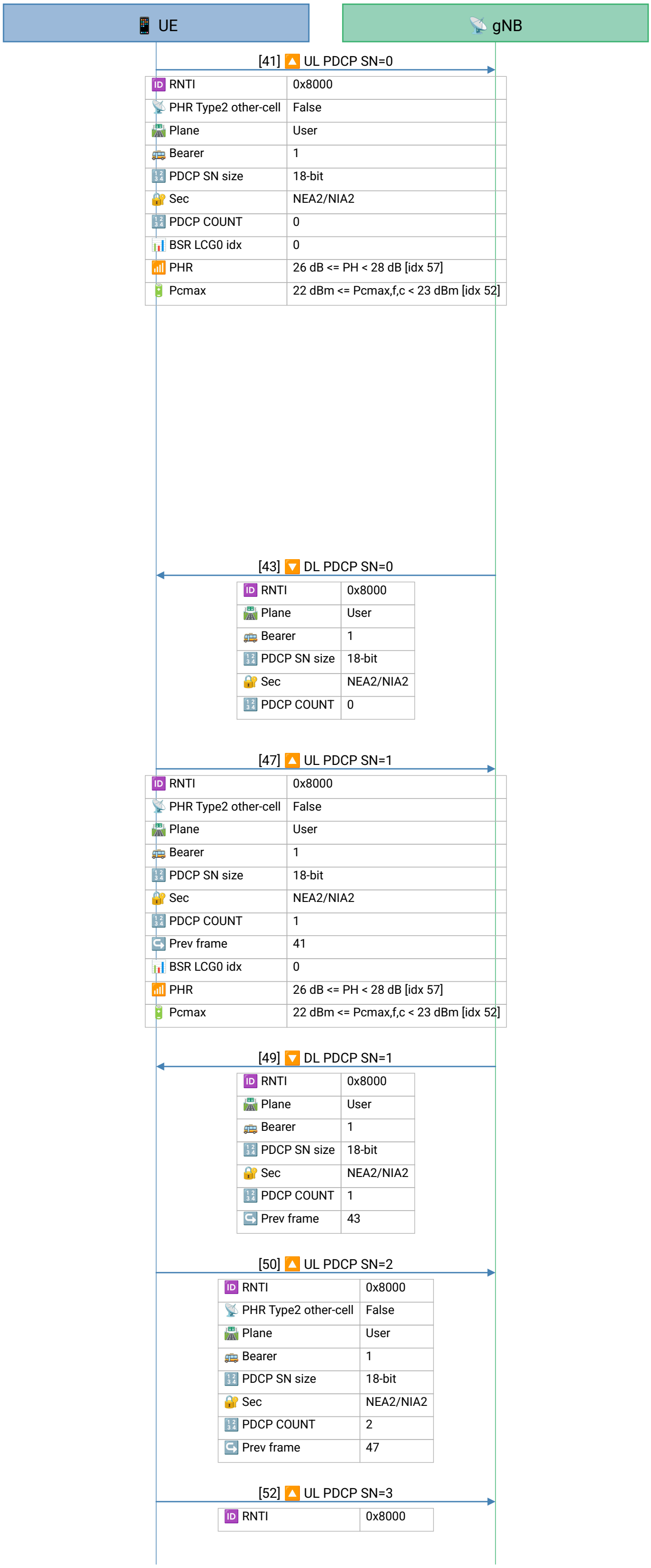
pdcp-nr-sequence-numbers-drb.annotated



[Frame 41] A second PDCP entity, three frames old, and a completely different header. `Seq Num: 0` again -- but nothing about this zero relates to the signaling bearer's. This is a new entity with its own `TX_NEXT`, created by the `RRCReconfiguration` of frame 38, and the header it produces is a different shape:

- First octet -- SRB 1: 4 reserved bits. DRB 1: a D/C bit plus 5 reserved bits.
- Sequence number -- SRB 1: 12-bit. DRB 1: 18-bit.
- MAC-I -- SRB 1: present, 4 bytes. DRB 1: not present.
- Header size -- SRB 1: 2 bytes. DRB 1: 3 bytes.

`1... .... = PDU Type: Data` PDU is the D/C bit, and it appears here for the first time in the capture. It exists only on the user plane, because only DRBs can carry a PDCP Control PDU -- a status report or ROHC feedback. Set to 1 it means "the rest of this is user data". `MAC-I Present: False` is the direct consequence of `integrityProtection is NOT present` in frame 38's `pdcp-Config`: this bearer is ciphered but not integrity protected, which is the normal configuration for user data.

One more field, and it is the subtle one. SDAP `header: Present, QFI: 2.` TS 38.323 §5.8 excludes the SDAP header from ciphering even though it sits inside the PDCP SDU, and the trace bears that out -- Wireshark's `Deciphered Data` for this frame starts `6000...`, the IPv6 version nibble, with the SDAP byte outside it. The QoS Flow Identifier has to stay readable so the receiver can route the packet without deciphering it.

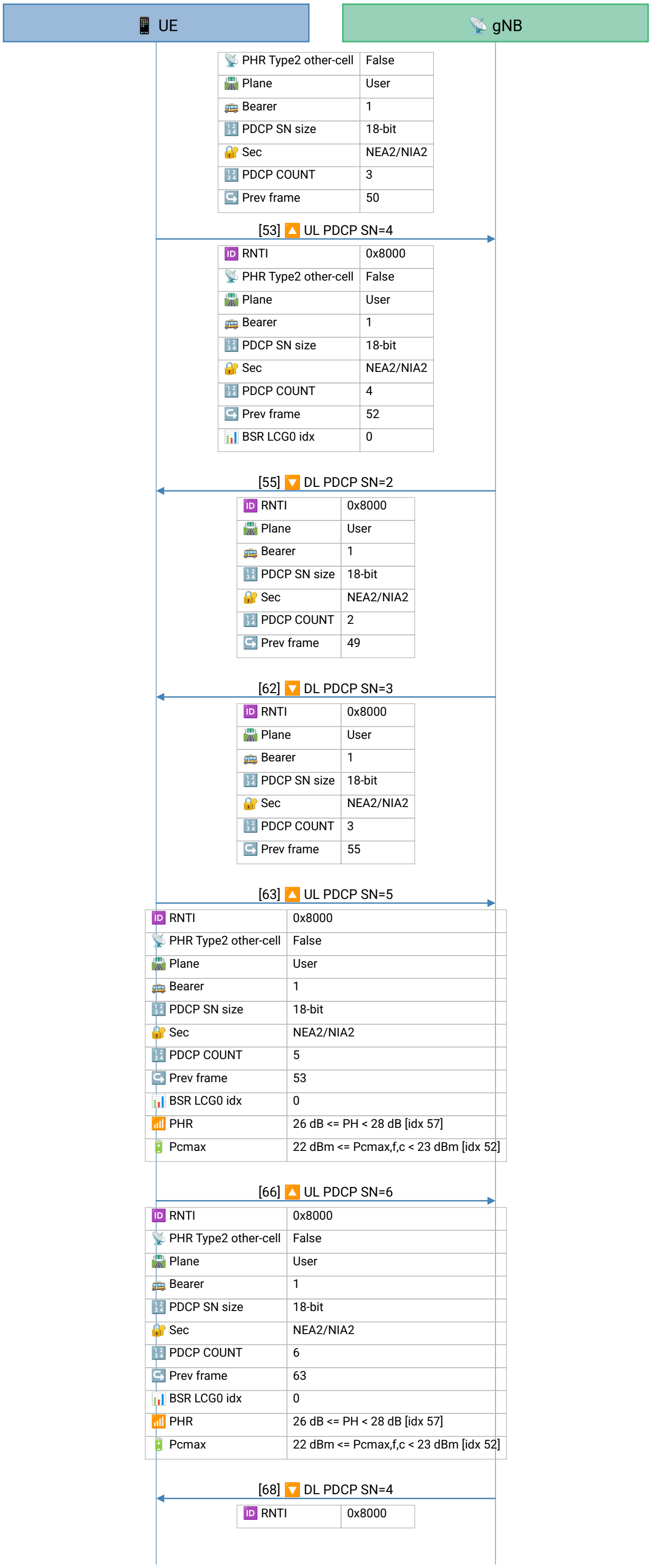
[Frame 43] The downlink entity opens its own count, and drops the SDAP header. Downlink `Seq Num: 0,` and SDAP `header: Not Present.` That is not an inconsistency: frame 38's `sdap-Config` said `sdap-HeaderUL: present (0)` and `sdap-HeaderDL: absent (1),` so the two directions genuinely carry different PDU layouts on the same bearer. The gNB already knows which flow it is sending on -- the QFI arrived from the UPF over N3 -- so repeating it downlink would tell the UE nothing new. Going up it is the gNB that has to be told, and with several flows able to share one bearer the header is the only thing that can name one.

[Frame 47] SN 1, and the payload changes character. The first two PDUs carried ICMPv6 neighbour-discovery traffic; from here the deciphered payload is TCP toward port 5060. PDCP neither knows nor cares -- the sequence number just goes to 1. What this trace's user plane actually turns out to be is the subject of the companion article on decrypting this capture.

[Frame 49] Downlink SN 1. The two counters are now at 1 and 1, and that coincidence will not last: the uplink reaches 17 while the downlink reaches 13.

[Frame 50] SN 2, and the first frame where the layer boundary earns its keep. On the wire this frame is three RLC PDUs -- a status report, a complete SDU, and the first segment of the next one -- but only one *complete* PDCP PDU arrives here, `Seq Num: 2, 64 bytes`. PDCP's counter advances once per SDU; RLC's activity in the same frame is invisible from this layer.

[Frame 52] SN 3, reassembled from three RLC segments into 1264 bytes. One PDCP sequence number, one COUNT, one keystream. This is the same event the RLC companion walks as frames 50 to 52; from up here it is a single number



ticking from 2 to 3.

[Frame 53] SN 4, reassembled to 514 bytes. Note that SN 3 and SN 4 were in flight at the same time -- frame 52 carried the tail of SN 3 and the head of SN 4 in one transport block. Interleaving like that is exactly why PDCP numbers SDUs before handing them down: the receiver can always put them back in order.

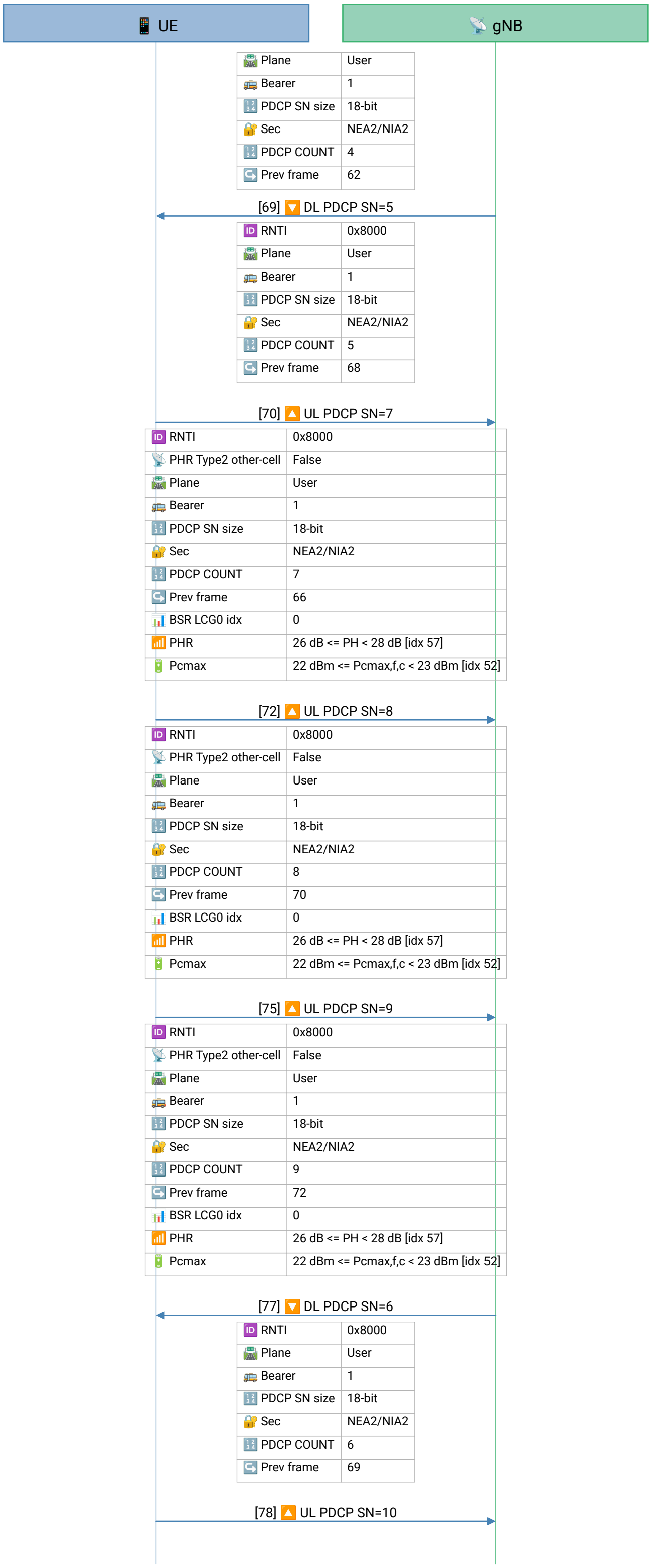
[Frame 55] Downlink SN 2, 2.782 ms after the uplink burst closed. 63 bytes. The downlink counter is at 2 while the uplink is at 4; independent directions, independent rates.

[Frame 62] Downlink SN 3, 791 bytes, after a 1203.007 ms pause. Whatever the application was waiting for, PDCP simply resumes at the next integer. There is no `discardTimer` pressure here either -- frame 38 set it to `infinity (15)`, so a queued SDU is never dropped for age.

[Frame 63] Uplink SN 5. 64 bytes. From here to SN 7 the uplink runs a steady sequence of small TCP segments, one PDCP PDU each.

[Frame 66] Uplink SN 6. 64 bytes.

[Frame 68] Downlink SN 4. 63 bytes.



[Frame 69] Downlink SN 5, 0.983 ms after SN 4. Two PDCP PDUs back to back in consecutive frames -- and note that the receiving side is entitled to see them out of order and reorder by sequence number. It never has to here.

[Frame 70] Uplink SN 7. 64 bytes, and the last of the small TCP segments.

[Frame 72] Uplink SN 8, and a third layer of encryption appears inside the PDCP plaintext. 108 bytes. The deciphered payload is no longer TCP: it is IPv6 next header 50 -- ESP, IPsec's Encapsulating Security Payload -- with SPI `0x00000100`. We deciphered the PDCP layer successfully and found more ciphertext underneath, protected by keys that were never on this radio link. It is the same lesson frame 12 of the signaling bearer taught with NAS, in a different place: PDCP protects one hop, and it is not the only thing protecting this traffic. The companion article on decrypting this capture follows that payload the rest of the way.

[Frame 75] Uplink SN 9 after a 1030.999 ms pause. 108 bytes, same SPI. Nothing in the PDCP header changed across a one-second gap.

[Frame 77] Downlink SN 6, 99 bytes, carrying ESP with SPI `0x0000fbeb` -- a different security association from the uplink's, which is normal for IPsec and has no bearing on PDCP.

[Frame 78] Uplink SN 10, 88 bytes, and the start of the session's largest uplink run. Over the next four frames the

📱 UE | 📡 gNB

🆔 RNTI	0x8000
📡 PHR Type2 other-cell	False
🛣️ Plane	User
🚌 Bearer	1
🔢 PDCP SN size	18-bit
🔐 Sec	NEA2/NIA2
🔢 PDCP COUNT	10
🔁 Prev frame	75

counter goes 10, 11, 12, 13 while RLC uses six transmissions to move them.

[80] 🔼 UL PDCP SN=11

🆔 RNTI	0x8000
📡 PHR Type2 other-cell	False
🛣️ Plane	User
🚌 Bearer	1
🔢 PDCP SN size	18-bit
🔐 Sec	NEA2/NIA2
🔢 PDCP COUNT	11
🔁 Prev frame	78

[Frame 80] Uplink SN 11, reassembled to 1288 bytes. Three RLC segments, one PDCP number.

[81] 🔼 UL PDCP SN=12

🆔 RNTI	0x8000
📡 PHR Type2 other-cell	False
🛣️ Plane	User
🚌 Bearer	1
🔢 PDCP SN size	18-bit
🔐 Sec	NEA2/NIA2
🔢 PDCP COUNT	12
🔁 Prev frame	80
📊 BSR LCG0 idx	0

[Frame 81] Uplink SN 12, reassembled to 988 bytes. Two RLC segments, one PDCP number.

[84] 🔽 DL PDCP SN=7

🆔 RNTI	0x8000
🛣️ Plane	User
🚌 Bearer	1
🔢 PDCP SN size	18-bit
🔐 Sec	NEA2/NIA2
🔢 PDCP COUNT	7
🔁 Prev frame	77

[Frame 84] Downlink SN 7, 87 bytes. The downlink is at 7 while the uplink has reached

12. If you ever need one image for why a merged PDCP timeline would be unreadable, it is this: two counters at different values, on the same bearer, at the same instant, each perfectly correct.

[90] 🔽 DL PDCP SN=8

🆔 RNTI	0x8000
🛣️ Plane	User
🚌 Bearer	1
🔢 PDCP SN size	18-bit
🔐 Sec	NEA2/NIA2
🔢 PDCP COUNT	8
🔁 Prev frame	84

[Frame 90] Downlink SN 8, 1223 bytes -- carried whole. No segmentation at all: the scheduler's grant happened to be large enough. Compare downlink SN 11 at frame 110, which is only slightly larger and gets cut in two. Segmentation is a property of the grant, and PDCP's numbering is unaffected either way.

[91] 🔼 UL PDCP SN=13

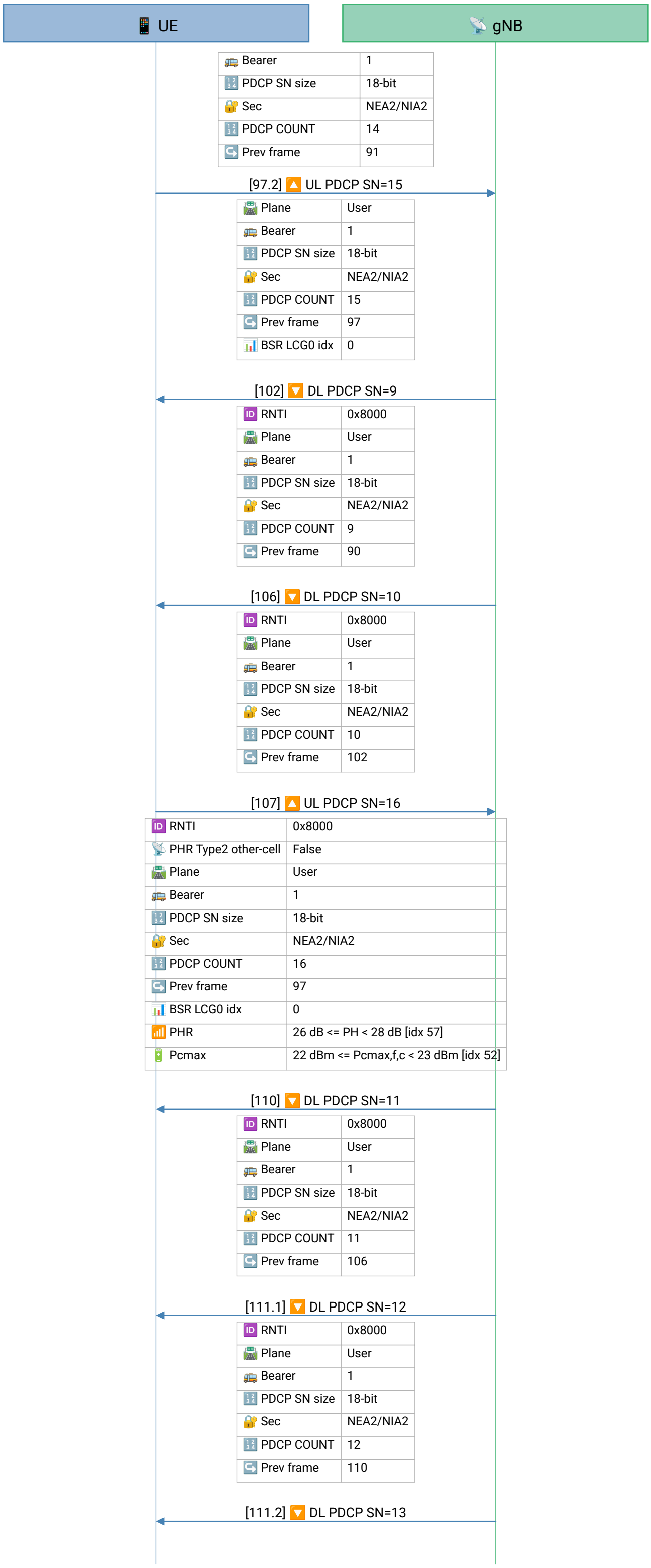
🆔 RNTI	0x8000
📡 PHR Type2 other-cell	False
🛣️ Plane	User
🚌 Bearer	1
🔢 PDCP SN size	18-bit
🔐 Sec	NEA2/NIA2
🔢 PDCP COUNT	13
🔁 Prev frame	81
📊 BSR LCG0 idx	0
📶 PHR	26 dB <= PH < 28 dB [idx 57]
🔋 Pcmax	22 dBm <= Pcmax,f,c < 23 dBm [idx 52]

[Frame 91] Uplink SN 13, 88 bytes.

[97.1] 🔼 UL PDCP SN=14

🆔 RNTI	0x8000
📡 PHR Type2 other-cell	False
🛣️ Plane	User

[Frame 97.1] Uplink SN 14, reassembled to 1288 bytes -- and the first of two PDCP PDUs in this single frame. The tail of SN 14 and the whole of SN 15 shared one transport block.



[Frame 97.2] Uplink SN 15, 332 bytes, in the same frame as SN 14. Two PDCP PDUs, two sequence numbers, two COUNT values, two separate keystreams -- arriving in one radio transmission. The diagram splits them into two arrows because they are two independent PDUs; the fact that MAC packed them together is a scheduling detail that PDCP never sees.

[Frame 102] Downlink SN 9, 87 bytes.

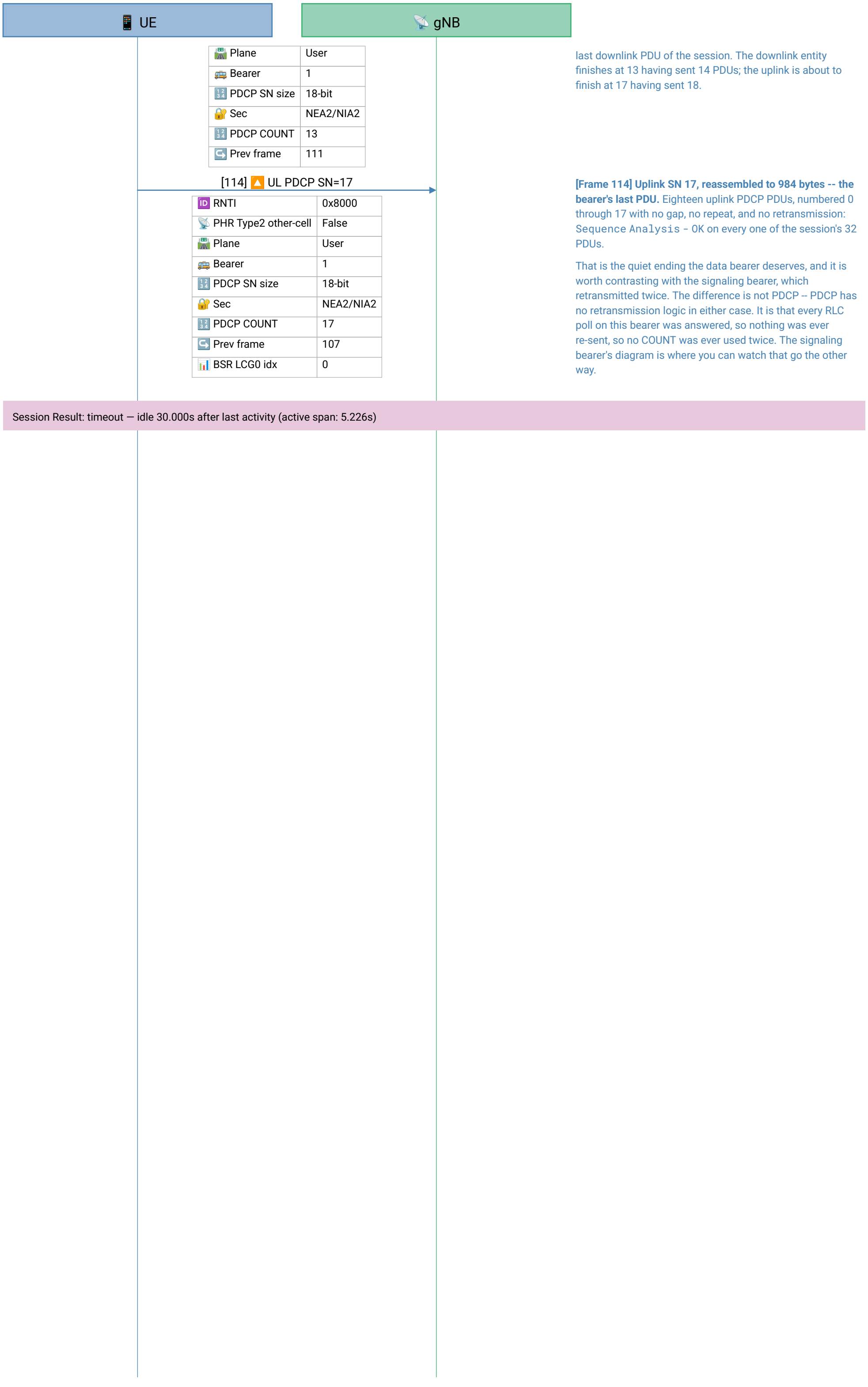
[Frame 106] Downlink SN 10, 587 bytes.

[Frame 107] Uplink SN 16, 88 bytes.

[Frame 110] Downlink SN 11, reassembled to 1499 bytes. Larger than the 1223 bytes that went whole at frame 90, and this one needed two RLC segments. The inner packet here uses IPv6 next header 44 -- a fragment header -- which is IP-layer fragmentation happening independently of, and invisibly to, both RLC's segmentation and PDCP's numbering. Three different mechanisms for splitting a payload, stacked, none aware of the others.

[Frame 111.1] Downlink SN 12, reassembled to 1499 bytes.

[Frame 111.2] Downlink SN 13, 575 bytes, in the same frame as SN 12 -- the downlink's mirror of frame 97, and the



last downlink PDU of the session. The downlink entity finishes at 13 having sent 14 PDUs; the uplink is about to finish at 17 having sent 18.

[Frame 114] Uplink SN 17, reassembled to 984 bytes -- the bearer's last PDU. Eighteen uplink PDCP PDUs, numbered 0 through 17 with no gap, no repeat, and no retransmission: `Sequence Analysis - OK` on every one of the session's 32 PDUs.

That is the quiet ending the data bearer deserves, and it is worth contrasting with the signaling bearer, which retransmitted twice. The difference is not PDCP -- PDCP has no retransmission logic in either case. It is that every RLC poll on this bearer was answered, so nothing was ever re-sent, so no COUNT was ever used twice. The signaling bearer's diagram is where you can watch that go the other way.

Session Result: timeout — idle 30.000s after last activity (active span: 5.226s)